An Interview with

JOSEPH F. TRAUB

OH 89

Conducted by William Aspray

on

12 October 1984

Columbia University (New York, NY)

Charles Babbage Institute
Center for the History of Information Processing
University of Minnesota, Minneapolis

Joseph F. Traub Interview
12 October 1984

Abstract

Traub discusses his academic contributions to computer science and mathematics at Bell Laboratories and at Carnegie-Mellon and Columbia Universities. He describes his work on iterative methods and the publication of his 1964 book, *Iterative Methods for the Solution of Equations.* He talks about his role in the development of computational complexity, out of an attempt beginning in the mid-1960s to construct a theory of operational algorithms. He also discusses algorithms he has constructed for the solution of linear systems and polynomials.

JOSEPH F. TRAUB INTERVIEW

DATE: 12 October 1984 INTERVIEWER: William Aspray

LOCATION: Columbia University (New York, NY)

ASPRAY: This is an interview with Professor Joseph Traub, October 12, 1984, at Columbia University. It is the second of in a series of interviews. The interviewer is William Aspray of The Babbage Institute.

Let's begin in this session by talking about your time at Bell Laboratories.

TRAUB: Well, let me welcome you first of all to Columbia University on Columbus Day of 1984. I hope an auspicious date. When I got to Bell Labs, I found I had this enormous hunger. It must have been something about the atmosphere of Bell Labs. An enormous hunger to do something really significant. I had no idea what that was going to be. One day somebody brought in a particular problem to look at, and that got me interested in iterative methods for solving equations. As I started looking at the literature, I was really surprised to find that literally thousands of papers had been written, but they were always about particular methods. There was simply no theory. No general theory of what it means for a method to be optimal in some sense. It seemed to me there was no scientific theory.

ASPRAY: I see.

TRAUB: It didn't come full flown, of course. That is, I realized there was no general theory and I started asking questions such as: if I added more memory to the method, what effect would that have. If I wanted higher order methods, would that be beneficial? I don't remember the moment when this happened. I had a crucial insight, namely: what really mattered was only the information that was available to the algorithm. The structure of the algorithm didn't matter. What mattered was the information. That is, Newton iteration uses a function and its first derivative. I was to prove later that any iteration that used a function and a first derivative could not have order of convergence greater than two. Therefore, the information itself determined the maximal order. In order to talk about

the complexity (I'll come back to what I mean by complexity), one had to know the maximal order. Furthermore, a method like the secant method one could view as using one new evaluation. What it knows about the problem that's being solved is what I mean by information--in this setting at least. I'll come back to information more generally later. Then it re-used one piece of information from an earlier time, namely the previous evaluation of f. It turns out the secant method makes optimal use of that information. That is, any other method with that much information (one new value of f, one old value of f) could not have order of convergence greater than the secant method. Again, it's not the speed of convergence that's really important, it's the complexity. That is, the minimal cost to derive an approximation of a certain quality. Over a period of several years, I realized that there were four, at least I introduced four, major classifications of iterations. They were: one point, which meant all the information was gathered at one point, for instance f, f', and f" evaluated at one point; one point with memory, an example is the secant method. So you have the new information at one point which you're allowed to remember. Multi-point iteration, which means that new information can be at several points, so it might be f at one point with f' at a different point.

ASPRAY: When you say "memory" here, you mean the previous calculation?

TRAUB: Exactly. The previous iteration.

ASPRAY: Yes.

TRAUB: These are all iterative methods. We're still talking only about non-linear equations. This was to be vastly generalized in the late seventies, early eighties. We're still very much involved in that process. But in the early sixties I started with a particular problem-solving non-linear equations.

ASPRAY: O.K.

TRAUB: The fourth class is multi-point with memory. One point, one point with memory, multi-point, multi-point with memory. Those are the four classifications. For each of these one could determine the maximal order of

convergence. The different cases were totally different. That is, for one point iteration the maximal order was equal to the number of evaluations. So, in some sense, it is what today we'd call the cardinality of the information. So if I have an f, an f', and an f'', then the maximal order was three. On the other hand, for multi-point iteration the maximal order was exponential in the number of evaluations. So with n evaluations we could construct an iteration of order $2^n - 1$.

ASPRAY: O.K.

TRAUB: Well, I guess the first time I ever presented these results was at the National ACM meeting in 1961. George Forsythe was in the chair. It was a session that made a very, very vivid impression on me. I had spoken to the American Physical Society about my thesis results, but this was really my maiden professional talk. There was Forsythe who was a man I'd heard of. He was later to become founding chairman of the Computer Science Department at Stanford; something we'll talk about later. He was in the chair and I remember he said, "I won't introduce the people. I will let the talks speak for themselves. I won't describe who the people are." I gave my talk and I was an arrogant young man. In those days I was very brash--very brash. But I really felt I'd done something very significant. I remember the first lines, I don't think it appears in the paper. But the first lines of the talk were, "People have worked on this problem for hundreds of years and it's never been a theory. Now I'm going to give you a theory." This was when I was 1961, I guess I was twenty-nine, and as I said it was my maiden talk. At least one person, Hans Maehly jumped up and said, "You're working on the wrong problem." But really what I was talking about was, what's the complexity of the problem, not how good was a particular algorithm. I will come back to that since it's a crucial distinction. It really made me feel good that I was working on problems that the greats have worked on. I'm not in the class of Newton or Fourier or Lagrange or Legendre or the various people who have been...I don't know if I mentioned Laguerre. But I'm not in that class. I know what I am and what I'm not. I'm not in that class. I'm not as good a scientist as those giants. Yet it made me feel enormously good to be working on something they'd worked on and do something that they had never done and in fact never tried to do.

ASPRAY: Yes.

TRAUB: In fact, that was to be a theme later. It always gave me pleasure to go to the library and read the work of those people in French, or in German, or in whatever language and have that sense of working on what these people did, and do things they hadn't been able to do. Not that they had tried, but it was that I was asking a new question.

ASPRAY: Yes.

TRAUB: Later, the method we were to create for polynomials zeroes, Jenkins-Traub (Jenkins was a student of mine) method and the Laguerre method are two of the main methods of solving polynomial equations. It still gives me pleasure to see in textbooks Laguerre's method with Jenkins and Traub. I have a very strong sense of the history.

So there I was saying I was creating a general theory. I presented talks in 1962 and in 1963. I started working on a long paper. I was just enormously involved for a period, I guess around...I joined Bell Labs in 1959. I can't tell you exactly, although possibly I could by looking at my notebooks. But either late 1959 or 1960 I started working on this. I simply buried myself in it. It was all that mattered for a while. I was never to have such a period again in my life. I was absolutely fascinated. I would often work until two in the morning. I remember one day my boss wanted to have lunch with me and somebody else. I begrudged him the time of that lunch because I just wanted to go back and keep working on this.

In about 1960 or 1961 I started writing a paper on these results. It was a hundred and forty pages long. I submitted it to the journal of the ACM. I remember Mario Juncosa sent me back a letter saying, "It's going to be a long time before I read this." Of course I was so naive and ignorant I didn't realize you don't send a hundred and forty page manuscript to a journal. Shortly before that, John Davis of Prentice Hall had written me, as I'm sure they wrote scores of people--asking, would I like to write a book. My initial reaction was "What do I know...a book?" I didn't know enough to write a book. I must have remembered this. Something was ignited, because now I had this manuscript. I went to Prentice Hall and said, "Yes, I would like to write a book on this." And they signed. I didn't want to put in

the effort to write a book until I had a contract; but they gave me a contract in 1962. Then I set about turning this into a book; it had to be perfect. Absolutely. I remember taking enormous pains with it. For instance, I remember doing the index took me two weeks. The copy editing of the galleys took the kind of time and energy that I know I never again funneled into such a matter. But it was very satisfying. In 1963 the manuscript went off to Prentice Hall, and the book appeared in 1964.

ASPRAY: Title?

TRAUB: The title was *Iterative Methods for the Solution of Equations.* It is a terrible title. I remember writing George Forsythe who was the series editor to ask about that as a title. He liked it. Looking back on it, it's a terrible title. Very non-descriptive. And the book is not about methods at all. The Russians are translating it. I wrote the Russian who was translating it, Sukharev. The Russians are also translating another one of my books, my 1980 book with Wozniakowski. They refer to the earlier book, the 1964 book, as starting optimal iteration theory. I just wrote to Sukharev a couple of days ago and said "No, I would like you to call the book, I would prefer that you call the book *Optimal Iteration Theory* because that's really what it's about." The history of that book is really gratifying. When the book came out I literally expected the world to say, "My god! Look at this beautiful piece of work," and to beat a path to my door. That didn't happen. That was disappointing because it seemed so obvious to me that it was so beautiful and so novel in that it really created a theory for classifying and understanding iteration.

ASPRAY: What kind of notice did it get?

TRAUB: Oh, the reviews were excellent, in fact. Although the author always feels nothing is good enough. Yet, fairly recently I looked back at a review by Alston Householder where I felt Householder hadn't been complimentary enough. He had said it isn't really iterative methods because I hadn't covered methods of polynomials, for example. I remember at the time thinking "Oh, that's because Householder was writing a book that included polynomials and it didn't seem quite fair." But I looked back at the review sometime later and thought, "What was I complaining about?" It was a very nice review. The other reviews were quite good as well. Looking back on it now, you have to

remember almost nothing was coming out in computer science yet. This was in 1964. So, you know, in a sense people didn't understand. I remember giving a talk at Toronto on it. Toronto was a strong theoretical department, and still the people didn't understand. That's because I didn't describe it adequately. People really didn't understand what the point was, why this emphasis on information was such a simplification, and why optimal iteration was interesting. It was amusing that in Toronto some years later, as computational complexity became fashionable, I was invited to give a talk again. I gave almost the same talk. This time people seemed to be much more receptive.

ASPRAY: Here is a question you may want to defer for a while: why was this important as part of the computer science? What was the contribution of this? How did it relate, say, to practical computing or to machinery that was available at the time, if at all?

TRAUB: I think the most important contribution was to create a classification for understanding iterative methods. It was just a hodge-podge. People would write a paper on the order of this and on the order of that. They didn't realize that order wasn't important. It was complexity and that was important and for that you needed maximal order. It was just a scattering of papers; there was no theory.

ASPRAY: Yes.

TRAUB: So I think the main contribution is not that I created a new algorithm--although I was to do that in other fields later. I was really trying to make a science of iterative methods. I told you the negative part of the reception, that the world didn't stop and say, "Wow!" On the other hand, probably nobody in the last twenty years has written a paper on that subject, that is, on the solution of scalar nonlinear equations, without referring to the book. It's absolutely the standard. Now if you look at numerical analysis books such as Ralston's book or Kronsjo's book, the classification, the words I introduced, is simply the standard one everybody uses. Nobody even says, "Traub introduced these." I introduced a bunch of now standard words in that book including the classification which is probably most important.

ASPRAY: Just so it gets down on tape someplace. What are some of these?

TRAUB: Oh, well, I mentioned one point iteration, multi-point iteration, one point iteration with memory.

ASPRAY: Multi-point with memory.

TRAUB: Multi-point. Well those four.

ASPRAY: Right.

TRAUB: One point, one point with iteration. Multi-point, multi-point with iteration. Also asymptotic error constant. I think there were probably several more.

ASPRAY: Yes.

TRAUB: It's had a wonderful history. It appeared in 1964. It was a fairly advanced monograph, not used for a text. Yet, it went through three printings and Prentice Hall kept it on its list until about 1980 or 1981. Sixteen or seventeen years! There was an initial surge when it sold to the libraries. But every year after that it sold the same amount. Year after year for about sixteen or seventeen years.

ASPRAY: That's quite impressive.

TRAUB: In some ways it's what I'm most proud of. That book still gives me pleasure. I remember meeting Jim Wilkinson in 1965 or 1966 at a meeting. Jim looked at me and said, "I didn't think you'd be so young." I was very flattered because here the book came out when I was thirty-two and it really created a theory. I was very flattered that Jim was surprised how young I was.

In 1982, Chelsea reprinted it. They came to me. I never thought of suggesting it. I wasn't particularity impressed until I saw their reprint list and saw I was practically the only living person on the list. There was Hilbert and others; everybody was dead on the list. I was really flattered. The Russians, Sukharev, who had translated our 1980 book said, "Well, we should have translated this a long time ago, and we're going to do it now." It's just about to come out now in a Russian edition, I guess in 1985, twenty-one years later. So it really is, as somebody once said to me, "a classic." It fits the definition of a classic. It just goes on. It's become the standard in my field. It's a book I'm very pleased with.

But as I say, in a way there wasn't that much early attention. I expected the world to sort of stop turning at this amazing piece of work, and it didn't; in a way I was discouraged. I've always been very stubborn scientifically. That is, it just doesn't matter how people react to my work. I've had lots of struggles. I cannot complain. Yes, some of the work I've done has not been immediately acclaimed, and it has never stopped me...I have a sense of what's worth doing and what isn't worth doing. On the one hand, I was disappointed and in fact went off and did some other things in the mid-sixties because there had not been much reaction to the book. But on the other hand, I always felt I'd been right. I jumped back into complexity when I saw the interest starting to appear in the rest of computer science. I've always had a sense that I knew what was deep and what wasn't. I've often had the feeling that some things I did that were relatively shallow were picked up. And the things I did that were really deep and really important weren't picked up or understood for a long time. Herb Simo n, who I regard as one of the really great men I've met...One of the great joys of computer science has been that one knows essentially everybody and there are some people I've met who I just admire enormously: Allen Newell, Herb Simon, Richard Karp, Donald Knuth, Henryk Wozniakowski, who's my closest colleague. Herb is one of the men who I admire enormously. He once said that if anything of his is understood for the first ten years after he's done it, he felt it really wasn't very deep or worth doing. I've really noticed almost a pattern. I don't know how typical it is of science that some of the things I've done which are really rather trivial got picked up and understood and the things that are really deep weren't. I think it says that even scientis ts often have to have things simple enough to understand to get a handle on. I'm sure it's that universal. But I've also noticed about myself I'm very stubborn. If I think this is right and this is deep and this is of permanent value, I will go ahead and hammer away at it--because I think I'm right.

ASPRAY: Come back to the book for a minute. Can you give me some indication of how the book served as a basis for the next line of research? Next area? It brought things together. Then what happened?

TRAUB: Ok, let me move into that. As I mentioned, I was somewhat discouraged that the world didn't stop. The invitations did not come flooding in to give talks, and so on. I went off and did some other things, of which the most significant certainly was the work on polynomial zero finding culminating in the Jenkins-Traub algorithm. Let me come back to that later and perhaps pursue the main line of the work on complexity.

ASPRAY: All right.

TRAUB: But I was enough discouraged by the reaction to turn to other areas. I could have pursued some of those ideas. I always have had a tendency to move around into different areas. This is really exciting, not staying in one area. So, partially because of the reaction of the book, partially because I have this penchant...The wonderful thing about computer science, which I cannot emphasize enough, is that almost any question I've ever asked, I've been the first one to ask. It's a wonderful thing. Unlike physics or math. I often say it was like being alive at the time of Euler or Gauss or Newton or whoever. Anything you ask you're the first one. Or maybe that's my taste. I hate to work in a field that's crowded where there are other people working. A mature field. I always seem to go and do things that nobody else has done. We're still doing that, surprisingly. I keep expecting that soon it'll become impossible. This is my twenty-fifth anniversary since my Ph.D. I got the Ph.D. in 1959. For twenty-five years that's what I've done. I don't like working in mature fields.

ASPRAY: Yes.

TRAUB: It's been the wonderful thing about computer science that you keep doing that. So I moved into other areas in the sixties. But I always had the special affection for this child of mine, this idea about the information. Look at the information; don't look at the algorithm. But nothing happened until 1968. I think that was the next significant date.

We're now going to pursue this track of complexity and then come back.

ASPRAY: Fine. Then we'll fill in the rest.

TRAUB: Exactly. It was a strange coincidence. In 1968, roughly, I went down to Princeton to a conference they were having, and I heard Al Borodin from the University of Toronto. Was then, is now. He gave a talk beginning with, "Isn't computational complexity a wonderful phrase. It's so sexy; it's so interesting." That's sort of what I remember. I was simply aghast because that's what I'd been doing. Computational complexity. I didn't have a word for it.

ASPRAY: Yes.

TRAUB: I realized that's exactly what I'd been doing. It's a theory of optimal algorithms. It turns out that the phrase "Computational Complexity" was invented in 1965. I think it was first used by Hartmanis and Stearns. But I hadn't been aware of it. In 1968, these people weren't aware of my work. There's been a sort of a split that we can come back to between essentially the continuous world, which is the world I live in, and the discrete world. I will try to distinguish what the difference is because it's very important in computer science and complexity theory. Well, that was simply an awakening that I now had the name for what I was doing. It was part of computational complexity, or that's what I felt. Now, what is complexity? Complexity is the minimal cost in some model of computation. It is the minimal cost of solving a problem. The cost depends on what you're interested in. It depends on the model of computation. It can be minimal time, minimal memory. In a VLSI chip it might be minimal area. It might be minimal length of program. It all depends on what you are interested in. But it is the minimal cost of solving a problem. It is an intrinsic property of the problem. It has nothing to do with how you solve it. It leads to a theory of optimal algorithms because an algorithm is optimal if it achieves the complexity of the problem. So it is an invariant that depends only on the problem. It's as central as the notion of work is in physics. Well, perhaps more central. It is an invariant as is the notion of work in physics. The notion of work, say against gravity, is such that the work you do depends only on the vertical distance that the object moves, not on the path. So it's inherent. But you can think of

the various ways of solving the problem as being like the various paths to get there. Anyway, complexity is invariant in that it talks about the intrinsic difficulties in problems. I'm talking about an absolutely fascinating question. Your background is mathematics?

ASPRAY: Yes.

TRAUB: It's a natural continuation of a stream of work going through Hilbert, Godel, Turing along the following lines. Hilbert said, "Is there a mechanical procedure for solving all problems of mathematics?" And Godel answered no. Essentially, that was what he said. And Turing said the decision problem is unsolvable. So, essentially there was a stream of very significant work going back to Hilbert. Dominating in some ways, and philosophically probably the most interesting thing for my taste in twentieth century mathematics is the issue of solvability. What a shock when it was shown that if the system is interesting, then there are theorems you can state but can't prove. Complexity seems to me to be a refinement of that. It seems to me in mathematics you have this dichotomy, either something is solvable or not. Complexity says, "Well, if it's solvable, it still may be intractable." That is, yes it's solvable, but the cost is so high that it's effectively unsolvable. So you have the notion of effectively solvable. But that really is something I find enormously attractive.

TAPE 1/SIDE 2

TRAUB: So just to repeat. It seems Hilbert posed the great question that was solved by Godel. I think of Hilbert, Godel, Turing as the great contributors there to unsolvability. Now computer science has essentially taken up the torch and said, "But now we have a refinement that says, if it's solvable, is it effectively solvable? How hard is it? And how hard is the problem, not the method?" So, if you have trouble solving a problem, is it because you're stupid and you haven't thought of a clever algorithm, or is the problem inherently that hard? I think it's one of the great problems.

Two asides here. I may have mentioned my wife, Pamela McCorduck, writes about artificial intelligence in her new

book which she has just delivered to her publisher called *The Universal Machine*. It is essentially a personal meditation on computers. The whole last part is the intellectual part of computers. She identifies two areas: artificial intelligence and complexity. She says those are the people who really turn her on, those dealing with the big issues. It's been wonderful for me, because she's now getting interest in complexity after having been interested in AI. In fact, I find also that the two really particularly interesting scientific philosophical issues are the nature of intelligence and the issue of what problems are solvable, what mathematically posed problems are solvable. Just to close a loop, I should say that after mathematics dealt with unsolvability, then computer science picked up the issue of complexity, now mathematicians are coming back in. Some of the world's leading mathematicians are now getting very interested in complexity issues. One example is Steven Smale, who for the last seven or eight years has been working on complexity, and some other very powerful mathematicians are starting to work on that. Gerard Debreu won the Nobel Prize in economics, and he says his major interest these days is the complexity of economics. I must say, I started a journal called the *Journal of Complexity*, which will publish complexity issues of interest to physicists, economists, statisticians, mathematicians, computer scientists. Well, back-tracking now, is it sort of clear what complexity is?

ASPRAY: Yes.

TRAUB: In 1968, because of Boroden's talk, I felt like the man who said, "Oh, I've been writing prose all my life." Oh, I've been doing complexity. To give you an example that it wasn't accepted yet, in 1970, I went to the University of Washington for a year of sabbatical from Bell Labs. I wrote my first proposal for the National Science Foundation, the mathematics branch, proposing work in two areas, complexity and parallel algorithms, i.e. algorithms for parallel computers. These were to become probably the two main research themes for the 70s and 80s. But it got turned down on the grounds that these were not interesting issues. I was very angry at the time. I had several exchanges with that particular person in charge of that program. He said later, "Why don't you apply to us again?" I reminded him of being turned down and being really angry about it. So, even in 1970 I couldn't get complexity research funded. But I still had that love for it and in 1970 or 1971 I wrote a review paper which appeared in SIAM *Journal of Computing* about iterative methods, using a little bit of new work and some new insights I had. Essentially, I wrote it because I wanted still to interest people in this issue. But nothing really significant happened until 1973. That is

when Henryk Wozniakowski sent me one or two papers from Warsaw with his results. He asked if he could visit Carnegie-Mellon where I was by then. I found the work very interesting and invited him to come. I think I invited him for a rather short amount of time because I didn't know the man, for three or four weeks, perhaps six weeks. That was to be for me absolutely a scientific turning point. Wozniakowski sits in that office right over there. We hope to convince him to, in fact, stay at Columbia and accept a full professorship here. He's my closest colleague and a man I admire enormously for his scientific creativity. He's a staggering mathematician; but I admire him also for his character. He's simply a marvelous human being.

It all started in 1973. I'm now answering that earlier question about what was to come. Fate can be strange. In 1973, a professor in Poland had shown him my 1964 book. He'd gotten interested in it and had put it into an abstract setting, an infinite dimensional setting. He devised what I would today call an "adversary principle", a mathematical machine essentially for dealing not with the scalar case as I had done, but with the abstract case.

ASPRAY: Yes.

TRAUB: He wrote two papers appearing eventually in about 1974 or 1975, which really settled a lot of the conjectures. I tend often to pose conjectures. He settled a number of those conjectures and really moved forward from the scalar case to the infinite dimensional case. I find it effective, particularly these days since my time is so tight, but even earlier I contributed mainly by being able to ask interesting questions. That's the way I interact with students and colleagues, by asking questions. I do that more these days than sitting down and actually proving things. So Henryk came then and every year he would come and visit Carnegie; or by this time I was spending my summers at Berkeley, so he would be at Berkeley. He would come and we would work together, still working on only one problem, the solution of non-linear equations. Then the next crucial point was in 1976. But before that, again I have to say how really remarkable that coming together was. Here was a man in Warsaw who I'd never heard of, who'd been handed a copy of my 1964 book, who'd written his Ph.D. thesis on it. Then we made contact in 1973. Then he was to become my main scientific partner.

Also, before coming to the events of 1976, I'll only mention that in the early-to-mid-seventies I had a student who I'd

met in Seattle named H. T. Kung, who was working with me. I'm often enchanted by the fact that, of those closest to me, Henryk is a Polish-Catholic, Kung is a Taiwanese Lutheran, and my wife is an Irish-English Episcopalian/Presbyterian. The fact is that I have so much more in common with these people than people who might be closer in any other way. Anyway, Kung, who was my student and who I am enormously proud of, today is the leading academician in the world in VLSI, systolic arrays, and so on, was working in this area, which by now we had given the name "analytic complexity", as opposed to discrete complexity being done elsewhere. This was analytic complexity because the tools were from analysis. We were still only working on nonlinear equations. I was always ambitious. Back when I was working on scalar equations, I wanted to do this abstractly. I wanted to do it for other areas. But I had no idea how to do it. The breakthrough came in 1976, and another student of mine named Arthur Werschulz, on the faculty of Fordham University and part of our research group at Columbia, gave a seminar where he used some of our ideas from nonlinear equations to work on problems in integration. My reaction was that integration is so inherently different from nonlinear equations and you can't do integration iteratively. My contribution was to say, there's got to be a general theory underlying all this. Because those problems are so different there must be a very general structure that we don't know that underlies this and lots of other things. Henryk and I were so interested in this. We always maintained long lists of our research problems. The flavor of the field is that there are always many, many, many more problems that we can solve than we have time to solve. Again it's because it's so new. Thus, we're always generating files of questions. But we were so interested in this one, that we called it the S question, or the S theory, which stood for Special theory. What theory underlies nonlinear equations on one hand, integration on the other, and hopefully many other things. We wrote several reports around 1978 and finally in 1980 wrote a book called *A General Theory of Optimal Algorithms*, which is a general abstract formulation.

Let me say with some hindsight that at that point there was some objection to me calling it a *general* theory of algorithms because it dealt essentially with this world that I worked in of continuous problems. I was very careful to say it wasn't *the* general theory, it was *a* general theory of optimal algorithms. Let me say again that I'm enormously proud of this book because it really was the first synthesis, the first step away from solving nonlinear equations. Now that it's four years later, I can see what the limitations are. In fact I could point out how we're removing them.

I should contrast this work with the numerical methods and numerical analysis on one hand and the rest of complexity on the other. Let me first tell you, now everything is done abstractly. The setting is abstract, with two important restrictions. The first is that it's a worst case setting. That is, you want to solve the problem for every element in some set of elements. This is as opposed to an average case setting where you want to know that on the average you're O.K. It's called mini-max in some fields. Secondly, we talked about the error in terms of a norm. So we assumed that the solution lay in a normed linear space. So the error is measured by a norm and the setting is the worst case. Those are the two restrictions. Otherwise, it was extremely general. We proved some amazing results in that book. Let me not go into detail other than that there was a very general attack on the complexity of continuous problems.

ASPRAY: All right.

TRAUB: Let me distinguish this work, in fact this whole flow of work that's still continuing, from the work being done in the rest of complexity. Because complexity is now a huge field. It is the central field of all of theoretical computer science. There are far more people working in the area in algorithms and complexity than working on other important areas such as verification, languages, logics, and so on. The largest number of workers and the most excitement for quite awhile has been in this area of algorithms and complexity. What distinguishes our particular approach is the following: people in the rest of complexity make the basic assumption that information is complete, exact, and free. Let me use, as an example, a typical NP-complete problem like the traveling salesman problem.

ASPRAY: O.K.

TRAUB: The assumption is that you know exactly the inter-city distances. The information is complete. Comp lete means that I really know what problem I'm solving. You assume that those numbers are given exactly. So the information is exact. You assume also that the information is free; you are simply presented with the inter-city distances. If you think about other problems, graph isomorphisms, matrix multiplication, it's always the same thing.

A good example is the matrix-multiplication problem; how many operations are necessary and sufficient to multiply two matrices. It is famous because the results are so astonishing. You can multiply a pair of two by two matrices, not with eight multiplications, but with seven. When that came out in 1968, it really shook the complexity field. It gave an enormous push to complexity. That was the result of Strassen. Now the question is how fast one can multiply two n by n matrices, and the best exponents are now about $n^{5/2}$.

ASPRAY: Right.

TRAUB: O.K. With matrix multiplication, what's the assumption? The assumption is that the input matrices are exactly given to you. You want to compute A x B, given A and given B. Given the data completely, you assume that the coefficients are exactly known and, furthermore, that they're given to you free. Nobody, by the way, actually states those assumptions. They are simply made implicitly. I talk to a lot to people in that part of the world, people like Dick Karp. I spend my summers at Berkeley, and Karp is the person I talk to particularly. Those assumptions are not made explicitly by people working in those areas. I only make them because I want to distinguish those assumptions from the assumptions that I make, which is that the information is partial. It is contaminated. It may be both. It might be partial, it might be contaminated, and it might cost you.

ASPRAY: Cost in the sense of computer time, for instance? Like any of those various things we talked about before as being associated with cost?

TRAUB: Right. For instance, needing to evaluate a function or evaluate a derivative. Let's look at integration as a typical example in this domain. Can you see the contrast? Throughout all the great work on P = NP, certain underlying assumptions are not stated explicitly. Information is complete. It's exact. It's free. The assumption we make is information is partial; it is contaminated; and it costs. I've already used traveling salesman and matrix multiplication as examples of the case where the information is complete. Consider integration. First of all, any problem you get in real practice, as opposed to a calculus course, you can't integrate symbolically, the anti-derivative does not exist in finite terms. You have to compute it numerically. Now, typically, when you're computing it

numerically, when you're approximating, you don't use the function itself. You use information about the function, like function values. Typically, you use some samples of the integrand.

ASPRAY: Yes.

TRAUB: Even if you have the function. But for most functions, you don't use the function in approximating the integral. You use something like function values. More generally you use linear functionals.

ASPRAY: Yes.

TRAUB: That means the information is partial. I don't even know what intergrand I've got. All I've got is some partial information on the intergrand. Now that's crucial, because that means that there are lots of intergrands which are indistinguishable with that partial information.

ASPRAY: That's right.

TRAUB: Secondly, typically function values aren't exact because they're either contaminated by computer error or they are the result of measurements. Because the integral really represents some physical quantity like the average energy of an earthquake. Often, measurements are contaminated. The information also costs me. Either I have to have sensors there that are getting the information or I have a cost computing function values. The information costs.

ASPRAY: I understand.

TRAUB: It is my assertion that almost any real problem, even mathematically posed problems like partial differential equations, where I don't know the starting conditions analytically, or I can't use them analytically, so I use samples of the starting conditions. Or any real world problem in economics or physics or engineering. Like statistics, the whole

discipline deals with the problems where the information is partial and contaminated. My ambition these days is to create a general theory--we are creating a general theory--called "Information-Based Complexity" for dealing with such problems.

We decided recently we have to come up with a different name for this field I've described in which the information is partial, contaminated and costs. We have used several names: analytic complexity, epsilon complexity; and to my mind the best name is "Information-Based Complexity" because information is central. If the information is partial, then you can work at what I call the information level rather than the algorithm level. If the information is complete, the only thing that gives you tools for dealing with complexity is the formal mathematical model, such as the Turing machine model. That's the only way you can get lower bounds. The big issue is lower bounds. Upper bounds on complexity are algorithms. If the algorithm costs so much, the complexity has to be less than or equal to that. But the interesting issue is the lower bound because that can only be delivered by a theorem that says there cannot be anything better. That's the deep one. In fact, John Hopcroft once said to me, "Upper bounds, that is computational simplicity; lower bounds is computational complexity."

ASPRAY: More precisely mathematically, least upper bounds?

TRAUB: Exactly. Least upper bounds are interesting and not delivered by algorithms. Typically, we don't know the least upper bound because then we know the complexity. For many problems, of course, all we know are some upper bounds. We're interested in the best current upper bounds. I think that's what you mean by least, you don't mean in the sense of analysis, of a sequence of subsets.

ASPRAY: Yes. I don't mean it in that way.

TRAUB: Right. That's exactly right. Those are interesting and useful. That's creating good algorithms. But the deep result is the lower bound. It's strange, you'd think that problems with partial information would be harder--these problems from optimal control, from scientific computation, or from partial differential equations. In fact, typically

these problems would be harder. However, using information that is partial gives us powerful arguments that are not available from problems with exact information. Therefore, our results tend to be independent of the detailed model, and we tend to know the complexity often much more accurately than people working with complete information. For example, the most interesting open problem with complete information is whether P = NP. That means we don't even know if these problems are exponential or polynomial. Whereas, for our kind of problems we often really know the complexity. That's because the fact the information is partial gives us a tool people don't have if the information is complete, simply because the information is complete. We think at the information level. What kind of questions are we interested in? Well, the first thing I should say is that if the information is partial or contaminated, you cannot solve the problem exactly. So the issue for problems where the information is complete and exact, when you can solve the problem exactly, is how much it costs. We have a different kind of problem. You can't solve the problem exactly. In fact, for problems from many disciplines, you can't solve problems exactly. That is because the information is partial and contaminated. You cannot solve the problem exactly; so you need some idea of approximation. We seek an E-approximation in a worst case, average, or probabilistic sense. The first question then is: I have certain information, what is the intrinsic uncertainty in the solution due to that information. Apart from the algorithm. There exists a quantity which makes everything possible. There exists a quantity, called the *radius of information*, which depends only on the information and how partial it is. It does not depend on the algorithm. By looking at the radius of information, I can tell you *a priori* whether you can compute an epsilon approximation. The main theorem about radius of information simply says that if the radius of information is less than epsilon, then I can compute an epsilon approximation; and it's if and only if. In fact, that expository paper I gave you discusses radius of information. So the first question is whether the information is strong enough to compute an epsilon approximation. If I can compute an epsilon approximation, what is the optimal information. For instance, where should I sample my integrand? Where should I place my sensors? Then one can ask what the complexity is of computing epsilon approximation? What is the optimal algorithm (that is the algorithm whose costs are as small as possible)? Those are the kinds of issues we're interested in. Those are the kinds of questions we ask.

We're interested in developing this field theoretically, by which I mean developing worst and average case models. Average case models with respect to very general measures on the space of problem elements. Then we're very

interested both in testing the models and showing the power of the theory by applying it to as many different disciplines as we can. Currently, we're very interested in vision, both robotic and computer vision. A student, David Lee, showed our work could be applied to vision. Say a robot is trying to see an object. The robot actually gets a finite number of the samples of the values of the surface. I'm talking about recovering a surface. That's a simple example of a vision problem. Most work in vision has been ad hoc. We are currently using our general attack to solve problems of vision. For us that's a good application. We're trying to do the same for many fields. Statistics is another one. It sounds almost arrogant to go into a discipline and want to make it a discipline. It's almost like my arrogance of 1961, saying I'm going to create a theory of iteration. But that really isn't the goal. All these disciplines which look so distinct, control theory, prediction, computer vision, estimation, statistics, scientific computation, are all the same in an important way. There are certain similarities because they all have partial information and there are certain questions which are discipline independent that we can answer. That in fact is the goal. Very ambitious. Just one more word on that. The nature of these fields is such that they tend not to talk to each other, and the methods are often *ad hoc*. For example, the criteria for Gaussian integration is you want to integrate polynomials of maximal degree exactly. That's Gaussian integration. Now if you're integrating a polynomial you don't need help. If you're integrating a function that isn't a polynomial, why should that be a good criteria? It isn't. The question is, what's the optimal algorithm? What has really always bothered me about all this, right back to 1959 or 1960 is the *ad hoc* nature of it all. Our belief is that we can create a science where the only thing you have to tell us mathematically is what problem are you solving, say by specifying it through an operator. So just specify the problem. Essentially that's all you're going to do.

TAPE 2/SIDE 1

TRAUB: The idea is to get rid of the ad hoc nature of all these different fields and simply specify the following. We wish to compute an element S operating on f where S is a linear or non-linear operator or transformation. f belongs to some set F. We want to compute an epsilon approximation in some setting according to some error criterion. Some approximation to S (f). Now the crucial assumption is that the information we are given is some other element N operating on f. What characterizes the field is that N is partial or contaminated. Therefore, you can only compute an

approximation to S (f). Everything else is to come out of the theory. The theory will tell you: what is the best information, what is the algorithm, what is the intrinsic complexity? Is adaptive information more powerful than non-adaptive? Is a linear algorithm optimal? Everything is to *come out* of the theory. Whereas, I find in all these fields people make all sorts of ad hoc assumptions such as *assuming* linear algorithms, without even realizing they're making *ad hoc* assumptions. That's the goal.

How do people react to this? My sense isn't that the world is rushing to adopt it; although there are some successes. I am a chaired professor in a great university and our research work is well-funded. There have been some excellent reviews. Our monograph immediately sold out, in translation, in the Soviet Union. The general theory is very new. The first book was done in 1980. The world is not rushing to embrace us. Now, what's the reaction of some people? Some people react when they see the word "information". Scientists are no different than other people. They know the word and have some connection they're eager to make. They might say, "Oh, that's information theory!" Actually I have people tell me that what I'm doing has all been done by information theory. I have to point out first that Shannon and Weaver really called their work "the mathematical field of communication", not "information theory" in this sense. It's communication theory. So they think it's been done by information theory. Or people say, "You can't use the word 'information'. Its technical use is reserved for the work of Shannon." I have to say, "He talks about 'information content'." I've actually had famous scientists say, "Your thinking must be sloppy because you're using words like 'information', and 'information' is reserved to Shannon." They're thinking of Shannon entropy. That's not information; it is information content, it's a number. So it's not the information of the message, it's a number that describes the content of the message. Furthermore, if you look at papers in many fields, people really do call something like this "information". Not "data". "Information" is the natural word. But you have no idea how many headaches have been created by my use of the word "information". Even though I was doing it since 1961. I started using the word then, but especially recently people have been really upset about my use of the word information. It's strange. I'm mentioning this to you in terms of history of science. It's strange the kind of things one can be attacked for. In fact my observation is that the space in which people can object to your work is an infinite-dimensional space. No matter what I've heard in the past and what questions I've answered in talks, there will always be some~p new question that I'm asked. So that's one real objection: how can you use the word

"information"? The information is what I know about a problem.

Another group that is upset about our work are some numerical analysts. We publish, in fact, in the best numerical analysis journals, *Numerische Mathematik*, for example. But some numerical analysts are very upset that we do not look at the details of a particular problem. One objects that we're simply trying to create a theory for the sake of creating a theory. He says "You and Henryk are so smart, but it's too bad you're trying to create a theory for the sake of creating a theory." That's one kind of objection. A second kind of objection is that you can't do things that generally. This from people who spend their lifetime working on one particular problem. It might be eigenvalues of a matrix or solving linear systems. They get very upset that we're including their work as part of a general theory. They claim that you must look at the details of that particular problem. It's true for many things such as for detailed rounding error analysis. But there are many things you can answer very generally. In about 1979 or 1980 was when Henryk and I said, "Well, it's really too abstract for some people who tend to work on finite dimensional problems, It's really too abstract. Let's work on a problem that we really are interested in understanding, linear systems, and that they'll understand." We wrote a paper on linear systems in 1980 but it did not have the desired effect. I actually had someone write a report attacking our paper. He was a referee for the journal of the ACM and said, "If this paper is published, I will insist on you publishing my attack on it." I think what in retrospect caused it, is that I was trying to write something they would understand. They didn't have to understand the general theory. Then they really got upset. I think it was so threatening that we could come into their turf from the outside and say that we could look at this more generally. I think that's what happened. It was clear these people didn't understand complexity. I asked the Editor to use referees who knew complexity. There was a four-year delay in publication.

My whole way of working is to try to have people come in and work on problems I'm interested in. In fact, I'm always saying, "My work's connected to statistics", or "My work is connected to control. My work is connected to...". I'm particularity happy when I find connections with economics because I really respect people like Arrow and Debreu, and Hurwicz who work in economics. I always loved that; but I really have a sense of scientists being upset about their turf, at least some people. One man who has spent his lifetime working in this area got extremely upset about this fairly innocent paper, which was essentially an application of our general theory to one particular problem.

ASPRAY: Let me get this title on tape. It's *Optimal Solution of Large Linear Systems* by Traub and Wozniakowski.

TRAUB: If you're interested, I've kept for historical interest in the file this thick stack of reports, letters, and so forth from one particular man who works in that area. His attacks kept changing. I hope I'm representing it fairly. One of the things he said is, "I'm not interested in all matrices in a class, I'm interested in a particular matrix." Of course, the whole nature of mathematics is such that you're always working with classes. That's what a theorem is. The hypothesis of a theorem describe the class. If you are working on the computer, you want a subroutine that solves a class of matrices, never just one. But if I answered one objection, he'd go to the next.

ASPRAY: It just doesn't make sense.

TRAUB: Yes. The kind of question I love here is, say, that you know that a matrix is, symmetric. Then how complex is the solution? Now let me add another property like positive definiteness. Then how hard is the solution? Then let me add the condition number is bounded by m. Now how hard is it? Well, it's going to keep getting easier. But how much easier? That seems to me a fascinating question. Which is, what is the worth of certain properties of an operator or of a matrix; that's an intrinsic. That seems to me of great mathematical interest. It has nothing to do with methods for solving them, but simply concerns the relation between the property and the complexity.

There are some theoretical computer scientists who don't understand what we're doing. People working on other areas of complexity. Part of the problem, I think, is my fault. My feeling has been that we should keep pushing the research frontiers. Consequently, our books are rather difficult. The 1980 book really needed a lot of mathematics, infinite dimensional mathematics essentially and analysis. For historical reasons, theoretical computer science comes out of logic and discrete mathematics. Part of it, people just don't understand. Well, part of it I think is just my fault. I remember once going to Oberwolfach. I figured these were the world's smartest people in the field. There were all these distinguished people in the audience. I went to theorem, proof, theorem, proof format and saw all these people's eyes glazing over. I realized that no matter how smart people are if it's outside their field, you have to be so

slow and develop just one or two ideas. So, partly it was my fault.

I'd never done expository papers. I've changed that. Hence this expository paper. Part of it is that people don't even realize that in so many problems the information is partial and so you just have to think differently. For some reason that I really don't understand, it's been true for twenty-five years that a lot of people just find it enormously hard to think at what I call the information level. To me it's absolutely natural. It's an enormous simplification. All I need to think about is the information that's available. There are so many algorithms. See, you have information, and then an algorithm is a mapping from the information into an approximate solution. Well, there are an enormous number of mappings and it's an enormous simplification just to think about information. All we are saying is that the total complexity is going to be the cost of the information plus the cost of the algorithm. If you look at the information, you can tell what are the optimal ways of combining the information. People really find those notions very hard--even very smart people. I've had long conversations with Steven Smale, who is interested in the subject. Even so, he and I had several sessions this summer where he kept saying, "Where's the mechanism. I want to see a mechanism. I want to see a model. Why don't you have a model?" Because we just permit arbitrary non-linear mappings here. We'll tell you what the optimal non-linear mapping is. And somebody as good as Smale was having trouble. Finally I said, "Oh, I understand." You see, the information approach was why Steve was having trouble understanding. If the information is partial, you can use that as the basis of your lower bound argument. You don't have to focus on the algorithm the way you do when the information is complete. That's just the identity map. Then all you have is the algorithm and you have to focus on that. I don't know if I convinced Steve. But people that smart have trouble with it. On the other hand, Debreu's reaction...This was before he won the Nobel prize. I'd been talking with him regularly for six or seven years. He said to me, "Any right-thinking economist would be interested in what you're doing." It's sort of interesting now--if you want to create a field, the very mixed reactions we get. I don't know how typical our experience is if you're trying to create a new field cutting across disciplines.

I should say in all fairness that, although some people are critical, we have received much gratifying response--in particular, *Zentralblatt fur Mathematik* reviewing our 1980 book said, "It's very difficult, but it simply raises the level of work by an order of magnitude and repays the work you put into it." We have no trouble publishing our

monographs. Perhaps it really simply does take some time. This new level of abstraction looking at the information level is the next level of abstraction of algorithms. It only exists when the information is partial and contaminated. That I guess brings us up to date. Perhaps I should go back and fill in some of the other scientific work.

ASPRAY: Good.

TRAUB: Let me backtrack to my earlier scientific work. I mentioned that after the 1964 book came out I turned to some other things. Let me mention just a couple of these. One was to create a high quality library of numerical mathematic routines. They were portable and carefully tested. That project is still running to this day. A second project arose because I found that as a scientist I wanted to be told about work going on anywhere in the world that fit my profile of interests. Yet, it was an accident as to which papers happened to be bound in certain journals at a certain time. I can pick up almost any journal and find something of interest. I always think there are so many things I'm missing. I would like to be notified of whatever is interesting anywhere in the world and be able to get the title, the title abstract, or the whole paper depending on my level of interest. Brown and I created a system called "the Mercury System" for the dissemination of reports within Bell Labs, which is used to this day. But I really created it because of my own needs. Because of what I learned there, I convinced the math society to start the Mathematical Off-print Service, which ran for a while. After a while they abandoned it because it was not financially feasible. I was very interested in how do you get out information that people need. It's sort of an accident in what journal things appear. Not entirely, because people do send things to certain journals. I so often found papers by accident, that I wanted more of a system for being made aware of what was going on around the world. We wrote a paper. In fact, there's a funny story associated with that.

John Pierce of Bell Labs was asked to write a piece for a special issue of Daedelus, I think called "Towards The Year 2000." Pierce, Stan Brown, and I wrote a paper on the future of scientific journals where we said the future lies in printing separates, not journals. Separates because then they can be disseminated by computer. What we overlooked, the one thing we didn't have in there, which I didn't see the significance of, was that everything could be sent electronically. We focused on the dissemination system. The paper was called "The Future of Scientific

Journals." I guess Daedelus felt that it wasn't held really far out enough. In fact, the Mercury System already created something like that at Bell Labs. So it wasn't in Daedelus. So we submitted it to *Science*, where it appeared.

The funny story is this. The reason Pierce was involved is that he had come to us with this invitation to participate with him in the Daedelus special issue. The paper was really written by Brown and myself and we were delighted to have Pierce join us. Pierce's contribution was to write: "As Mark Twain said about the weather, everybody talks about it but nobody does anything about it." That was really his contribution. That was it. The reviewer for *Science* liked it, but said the reference is wrong; that it's widely believed to be due to Mark Twain, but isn't. Instead it's due to x, and I don't remember x. The funny part of the story is that somebody came to me after the article appeared and said, "John Pierce really wrote that article, didn't he? I could just detect his style."

Another side activity was the numerical mathematics library. By this time I was heading a research group, but I thought it was a useful thing to do for Bell Labs, to have such portable carefully tested routines. That's now a big area by the way: mathematical software, and other kinds of software. There is a journal called *Transactions of Mathematical Software*. But, typically, I got involved when almost nobody else was doing it. You really have to think very carefully about testing. It also has to be portable. Then you have to distribute it to people who have been using junk for software. Then I got out of the field. I've done that often. I like to move in to fields early. That's what's exciting. Then I get on to other things.

For me the really significant thing was work on finding what is now called the Jenkins-Traub algorithm. It is a method for solving polynomial zeroes. Even then, it wasn't so much that I was interested in polynomial zeroes. What I was interested in was for how general a class of non-linear equations one could get a globally convergent method. That was what we were interested in. That is, for the general problem $f = 0$, you can't get global convergence; you can only get methods that are locally convergent. But I got an idea of how to get a method that worked very generally for polynomial zeroes. I really got very fond of those methods. They are very beautiful. The method we finally came up with is nice, but the crucial thing is that I suggested to Michael Jenkins, who was a student at Stanford while I was a visiting professor out there, that he build a piece of high quality, carefully tested, portable software to implement the

algorithm. Today that method, the Jenkins-Traub algorithm is the most widely used method of solving polynomial equations with real complex coefficients. But I'm convinced that it wasn't just the algorithm. We had a very fine piece of software. A piece of paper may be read by a few people; but if you have a good piece of software, a good program, it can be used at thousands of installations. That turned out to be crucial. It's a piece of work I'm particularly happy with. It is a three-stage algorithm. It consists of three nice ideas, and that's it. I moved on. I did some consulting, and people asked me to give a talk on that method. I said, "That's work that's two or three years old. I want to talk about what I'm doing now." I tended never to exploit in some sense what I had done. I always wanted to move on to whatever was the next piece of work. Yet that is one of the things I'm particularly proud of because it's so beautiful. I think the other favorable characteristic that work had is that many of the great mathematicians and physicists worked on the polynomial zeros. Newton and Laguerre are certainly two giants, but there were a number of others. I guess Fourier had a method involving that problem. It really gave me pleasure to think I had come up with something better than what those people had invented.

Then, in the seventies, in addition to what I now call "information-based complexity", which I regard as my main current scientific thrust, I kept creating algorithms.

One of these is something called the Shaw-Traub algorithm. It involves an almost trivial problem. The problem dates back to the work of Horner in the early nineteenth century, which is itself a very interesting story. As you probably know, Horner was interested in finding polynomial zeroes, and his method depended on shifting polynomials, that is computing P (x + h) as a polynomial in x. It was a shifting method. The way he did that was through a Taylor expansion. The Taylor series expansion shows that you can do that by computing all the derivatives of the polynomials at the point h. He had developed a very beautiful algorithm. If you just wanted to evaluate a polynomial, that's also called Horner's method. You can do that with n multiplications and n additions rather than the obvious way. That's sometimes called Horner's method, and also called synthetic division. But you could get all the derivatives in $n^{2/}2$ multiplications, and $n^{2/}2$ additions. Very elegant algorithms. He tried to publish it in the *Proceedings of the Royal Society*. There was a certain amount of flack because it was such a simple problem. I think it was in Cajori I read an article about the history of Horner's method as part of a lecture on how you evaluate a

polynomial. If you do it the most obvious way, say you want to evaluate a polynomial at point x, you form x^2, x^3, x^4,...x^n and then multiply by the coefficients. That does not minimize the number of operations. One of the first pieces of work in complexity was a paper in 1954 by Ostrowski where he asked essentially whether Horner's role is optimal. Because Horner beats the obvious method. Horner uses n multiplications and n additions and beats the obvious method. Ostrowski gave the paper a strange title, "Two Abstract Problems in Algebra."

ASPRAY: Yes.

TRAUB: He asked the question, is Horner's rule optimal? He didn't solve it. He asked it. This is what so often happens in science, and I really believe this. The crucial step is not the answer to the question. You've got to ask the right question. That was a very significant question. Knuth has a description in volume two, of the history of Horner and Ostrowski asking that question. Well, it finally was resolved that Horner's rule is optimal with respect to the a number of arithmetic operations for evaluating a polynomial. So, I was giving a class on numerical algorithms in the early 1970s at Carnegie. I told the class about evaluating an algorithm. Then I showed them Horner's rule for computing all the derivatives of a polynomial in $n^{2/}2$ multiplications, $n^{2/}2$ additions, which is not the obvious method. I told them that I guessed it was optimal. A student in my class, Mary Shaw, showed me she could do it faster. Then we started working on it. We finally showed that you could compute all the derivatives of a polynomial, if you are permitted division...

TAPE 2/SIDE 2

TRAUB: So I conjectured that Horner's method of computing, for computing all the derivatives is optimal, and Mary showed me that we could do it faster. We worked on it. We finally got it down to the point where you could do it with just 3^n multiplications and divisions, instead of a quadratic number of multiplications. There was still the same number of additions. So, it was linear in the number of multiplications and additions, but still quadratic in the number of additions; and it was just as simple, if not simpler, than Horner's method. That was interesting history because it occurred by my making the conjecture, Mary saying that it is not true, and then her showing there was something

just as simple that could do it in a linear number of multiplications and divisions. Now I don't think that's significant in terms of practice. How often do you compute all the derivatives of a polynomial? But there are still interesting complexity issues even for that simple problem. For instance, the number of additions needed; I gave you as many multiplications and divisions as you want; say they're free. I conjecture you would still need a non-linear number of additions. That's open. So even for that simple problem there are open issues. Mary Shaw is now a tenured professor at Carnegie-Mellon University in software systems.

The way another algorithm that I'm very fond of arose is as follows. You take the following algebraic function. You have a polynomial in two variables, say P (w,z) = 0. You want to compute w as a function of z, an example is the square root of a polynomial of an algebraic function. A simple example would be W squared minus a polynomial equals zero. Then you're computing the square root of a polynomial if you're computing square roots, There have been whole books written on algebraic functions. What Kung and I showed, a result I'm very fond of, is that the classical result is very inefficient if you want to compute the expansion of an algebraic function. How hard is that? It's exponential in the number of terms. We showed you can compute the first n terms of any algebraic expansion at a cost of n log n, which is the same cost as it is to just multiply together two nth degree polynomials. So the algebraic function expansion which looks pretty hard is in fact as easy as multiplying polynomials of degree n. This is a very surprising result, one I'm very fond of. Now, that has a lot of characteristics that are typical. The problem goes back to Newton. Newton first invented Newton iteration for solving f(z) = 0, a function of one variable. He did it, by the way, just for a cubic; he never did it in general. He just illustrated it for a cubic. The next problem he works on is algebraic function expansion. He did it in a rather awkward way. His method got you one coefficient at a time, in the expansion. It turns out Newton didn't realize that Newton iteration was quadratically convergent. It turned out what you could do was a form of Newton iteration, not on numbers but on power series working over valuation fields. You could double the number of coefficients, not the number of digits, but the number of coefficients at each step. So what we did to derive that result was that we used Newton iteration on power series. That was very satisfying. We did something Newton had overlooked and it solved the problem that Newton and lots of people since then have worked on. The other part I liked is that it wasn't just for functions where there are power series of expansions. In general, you cannot expand in powers, but you have to expand in fractional powers. That is the singular case. An

example of that, is just $w^2 - z = 0$, where the solution is equal to $z^{1/2}$.

ASPRAY: Yes.

TRAUB: We could, by looking at what is sometimes called the Pusieux diagram or Newton diagram. We gave an algorithmic version. It was well known in algebraic function theory or in algebraic geometry how to do it mathematically. We analyzed the algorithm and showed how to regularize the problem. Once we had a regular problem we applied iteration. We finally proved that for every algebraic function you could always compute the first n terms at cost n log n. I've already made the connection with history. Again, I always have that sense of connection with earlier work; but the way the result actually came about is that my student, H. T. Kung, got the idea of applying Newton iteration on power series; so you are computing coefficients rather than numbers. He gave a seminar and he said that this idea always works. I immediately came up with an example where it didn't work. The question was what is the domain in which Newton iteration works and were led to algebraic functions by that result.

There's one other algorithm I'm rather fond of. That's today called the Brent-Traub algorithm. I should mention that, in general, through the early to mid-sixties, I worked by myself on optimal iteration theory. That was on optimal iteration theory. Starting then, I've always worked jointly with other people. I've had a sequence of wonderful colleagues. That's one of the wonderful things about science. My wife Pamela (McCorduck) says for her it's very lonely; she has to work by herself. She's only ever had one co-author Ed Feigenbaum in one of her books. She rather envies the fact that I'm always working with colleagues. I've had a sequence of very successful collaborations. The most important being with Wozniakowski now for eleven years. We've written two books together and we have written I don't know how many papers together. I'd have to count to see how many collaborators I've had. H. T. Kung is the most important after Wozniakowski. I think that's one reason I've been able to do as much science as I have, in addition to building institutions, because I've had these wonderful collaborators. So let me talk about one more collaboration, that's with Richard Brent who holds the chair of computer science at Australia National University. Our collaboration netted a small result I'm very fond of. It answers the following question. Brent had shown that you could compute to the qth power of a polynomial where q can be anything, not just an integer, at cost

no more than a single multiplication. So computing to the two-millionth power is no harder than doing one multiplication. That's a rather nice result. The trick is to use the exponential and the logarithm. He shows that you can compute the exponential and the logarithm as fast as you can multiply polynomials. Bang! The question that was raised in my mind concerned what happens when you take the next more complicated kind of operation after multiplication, namely composition. There had been some nice work of Brent and Kung earlier showing how fast you could do composition. The classical results go back to Fourier and others. They showed how to do it fast. The question that intrigued me was whether the same thing could work for composition; how fast could you compute the qth composite? What made the problem attractive was there was no function like a logarithm. The thing about a logarithm is you reduce multiplication to addition, and that's really what makes the process work.

ASPRAY: Right.

TRAUB: But there is no function for composition that reduces it, to say, multiplication. But there is a transformation on a power series such that composition can be reduced to multiplication. It's a transformation that goes back to Schroeder in 1870. Using that, we were able to develop algorithms by showing that Schroeder's transformation could be computed fast. We were able to show that the qth composite was no harder than a single composition, and furthermore, just like the multiplicative case, q can be essentially anything. So you can compute the one-half composite, which is almost like a square root. That is the function which when composed with itself is equal to the given function. Well, that can be done fast. That's now known as the Brent-Traub algorithm. I'm not sure there are any applications to that. But it's part of my interest in how fast can you do the basic calculations of mathematics. In fact, Brent, Kung and I, between us, developed fast algorithms for all the basic operations on power series. We wrote a series of papers; always working in pairs, we pretty much cleaned up that area. There are still some nice open problems. I'll mention one open problem for you to do on the plane back: prove that for any normal computation composition is harder than multiplication. One certainly believes that composition must be more difficult than the operation of multiplication. That trivial little problem is open, and no one has ever made any progress on it. Just prove that the complexity of composition is greater than the complexity of multiplication. Now the best algorithm known for composition of course has greater complexity than the best algorithm known for multiplication. That just

proves that multiplication is not harder than composition. Problems as simple as that are one of the things that make complexity so much fun. The most trivial kinds of problems are still open.

Let me just end this session with an insight I had recently. I looked at the twenty-five years since my Ph.D. and found I'd really done two kinds of things. I do science, and I build things. I build institutions. I change institutions. Sometimes it's departments. Sometimes I try to cause major changes in universities. I start journals. I'm always building institutions in science. It's unusual to do both. So there are people who are scientists and not interested in building; and people who build things who are not scientists. I ask myself the question, why is it that I have these two different skills? The gift for research I didn't even realize I had until I started my Ph.D. thesis and then especially in the 1960s at Bell Labs I suddenly realized that I had this gift for asking questions, lots and lots of questions, some of which were interesting. It wasn't until I got to Carnegie-Mellon in the early seventies that I realized I had this other gift, the gift of building institutions. Usually when I complete building something it sort of stays that way. It goes along fine without me; but it usually doesn't change much anymore. I really enjoy the creative act of building. They seem very different abilities. I suddenly realized only recently what they had in common, that I tend to work in situations where there is a lot of ambiguity. I see all these different methods and there's lots of ambiguity and uncertainty. Then with institutions it's the same. There is all this ambiguity and uncertainty. I seem to be able to put a structure in it. I deal with the ambiguity by putting on as simple as possible a structure. I come up with a simple structure dealing with the ambiguity, and I do that both with organizations and in science. I suddenly realized how the two things which seem so different are really two aspects of the same kind of need, which is to deal with ambiguity and uncertainty by putting on them a structure which I can understand. I have to put on something that's simple enough so that I can really manage it and understand it, and that's the drive on both sides.

ASPRAY: Can you give me an example on the administrative side?

TRAUB: Yes. When I got to Carnegie the situation was the following. The first chairman had been Alan Perlis. From what Allan Newell told me, he just intellectually dominated that department. In fact when he left they felt that might be the end of Carnegie as a great department. Then I came into this environment. I had never even been in any

genuine graduate department. I'd been at City College, which is an undergraduate college. At Columbia, I'd been a student under the Committee in Applied Mathematics because there was no department. I'd been at Bell Labs. I'd been a visitor at several institutions, but I'd never been in a computer science department. Even when I was at the University of Washington there was not yet a department of computer science. I had never really been in a computer science department, and here I was now head of one of the major computer science departments. I didn't think that was a disadvantage because it left me absolutely open-minded. There was already in some ways a rather large department. In 1971, it was small in terms of having only seven faculty; but it had a well known Ph.D. program and several million dollars in research. Equipment, facility, staff, and how was I going to understand that department? I created a mechanism. One thing I did was to convene a committee of students and faculty to review the Ph.D. program. I chaired that committee in order to really understand it completely. One of the mechanisms I created was the Black Friday review. I will explain in a minute where "Black Friday" comes from. It's still tradition at Carnegie. They just sent me that [Black Friday] cup. It shows the number of students from each of those classes who flunked out of that program on Black Friday--as many as eight in one year. As you see, it's a decreasing sequence. The reason is that there was more time for students in earlier classes to flunk out; that's why the numbers go down. Don't they go down?

ASPRAY: Well, roughly. There are a couple of aberrations.

TRAUB: Black Friday is a wonderful mechanism. We do it here at Columbia and I think Stanford adopted it. But I created it so I could understand the department at Carnegie Mellon. Black Friday is when the entire faculty sits down and reviews, twice a year in December and May, at the end of the Fall and Spring semesters, every single Ph.D. student. There is a discussion of every single student. Then we make a decision whether the student is doing fine, or whether the student will be asked to leave the program, or whether he's not doing pretty well and what he should do this to rectify that. The student is told this in a letter. It has all sorts of benefits. Every one on the faculty knows every student. Every student is told twice a year where he stands. It leaves a paper trail of all the earlier decisions, so we don't forget from semester to semester. It's a wonderful management tool. It acts as a forcing function to make us know the students. Black Friday is now widely known in computer science. The reason for the name "Black

Friday" is that shortly after we got to Pittsburgh, on a Saturday night (there's not much to do on Saturday nights in Pittsburgh) we were watching television, and there was something called "Chilly Billy's Mystery Theatre." He had a show on called "Black Sunday", a horror movie. It made a certain impression and a couple of days later, when we talked about creating this mechanism, this Ph.D. review committee, we decided to have this review, and the first one turned out to be scheduled on Friday. I referred to it as the Black Friday review. The name simply caught on. It's due to this horror movie. Again, with that mechanism it meant I would hear a review of every student in the department--no matter how long they had been there. It's the kind of thing I do; a very simple mechanism that has all sorts of side effects. Yet it's a mechanism which enables me to manage the department and understand it.

ASPRAY: I see. I think that illustrates it. You have a Knuth story to tell me.

TRAUB: Yes, it's brought out by your telling me that Turing liked to recreate his own results. I told you earlier of the work I'd done with Richard Brent on composition which Knuth reports in his book. Knuth was just revising volume two. He called me one Sunday and said that Richard Brent had come by some time earlier. Brent had been a Ph.D. student at Stanford, and had told Knuth about this result. Don (Knuth) had been interested and I guess had reconstructed how you would do composition fast, this repeated fast self-composition. He wanted to check with me to see if he had gotten it right. I said, "Well, interesting you should call now, because we had just written it up. It's just being typed and I can send it to you in a week." Don said, "In a week I'll be doing something else. "I'm interested in this right now, and I want to write it up now." We spent about an hour discussing the idea so that Don could write it up and move on to the next thing. What really struck me about that is that in my way of working I never want to do something somebody else has done. I would never do that. I would never waste my time trying to do something I knew somebody else has done. And it impressed me that Don had the insight and energy to reconstruct this result. He had done it correctly. He had some interesting insights. It also struck me how different we were. I would not want to reconstruct somebody's else's work, whereas Don went off with the bare idea and did it himself.

END OF INTERVIEW